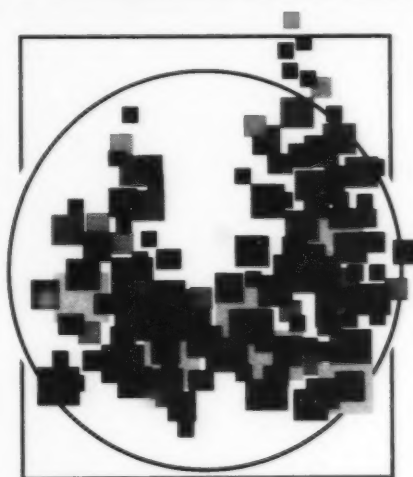




CENTRE FOR
SYSTEMS
SCIENCE

92-05739/

**Parallel Algorithms for Line Detection
on a 1xN Array Processor**

Ze-Nian Li, Frank Tong and Robert Laughlin

CSS-IS TR 90-06

July 1990

Parallel Algorithms for Line Detection on a $1 \times N$ Array Processor

Ze-Nian Li Frank Tong Robert G. Laughlin

School of Computing Science

Simon Fraser University

Burnaby B.C. V5A 1S6

Canada

Abstract

This paper describes two algorithms that compute the Hough transform for straight lines on $N \times N$ images using $1 \times N$ processing arrays. The algorithms are developed for $1 \times N$ array processors and have been implemented on the AIS-5000 parallel vision computer. The complexity of both algorithms is $O(N + P)$ on an $N \times N$ mesh and $O(N^2 + PN)$ on the $1 \times N$ array processor, where P is the number of angles polled which determines the θ -resolution of the Hough space. The actual execution time compares well with the known optimal $O(N + P)$ algorithms for $N \times N$ mesh arrays.

Index Terms Array Processor, Hough Transform, Line Detection, Parallel Processing.

1 Introduction

Many computer vision applications require efficient algorithms and special-purpose computer architectures in order to extract information at a sufficiently fast rate. This is particularly true in applications where the images are changing over time, such as vehicular guidance, robot vision, and motion detection. The Hough transform is one method of detecting objects in images that conform to a particular parameterized class [6]. In order to compute the Hough transform in a reasonable amount of time, increasing use is being made of parallel architectures and algorithms tailored for them [1, 2, 4, 5, 9, 10]. Somewhat less work has been reported on the actual implementation of algorithms that compute the Hough transform on real hardware parallel machines. In this paper we present two parallel algorithms for the Hough transform. Both have been successfully implemented on the AIS-5000 $1 \times N$ array vision computer. One of the algorithms is based on a published algorithm [2] for an $N \times N$ mesh array that has been optimized for the AIS-5000's $1 \times N$ architecture. The other algorithm has been developed by the authors. Both algorithms compare favourably with published benchmark results for the Hough transform on parallel machines [7].

One configuration for $1 \times N$ processor-array machines consists of N 1-bit processors arranged in a row with a column of memory for each processor. Many operations on $N \times N$ images can be accomplished in $O(N)$ steps, since an entire row can be processed in constant time. $1 \times N$ array machines have the advantage of a much simplified architecture and far fewer processing elements when compared to $N \times N$ array machines. Another advantage of this architecture is that operations can be carried out across an arbitrary distance within a column. For example, a pixel can be shifted vertically by any amount within a column of memory with one operation. The same vertical image shift in a typical $N \times N$ mesh array requires $O(M)$ operations since processors are only connected to their four nearest neighbours and moving each pixel requires M operations.

The $O(N^2 + PN)$ complexity¹ of the algorithms is comparable to an $O(N + P)$ complexity for the Hough transform on $N \times N$ mesh array architectures [2, 5]. Both algorithms require copies of the edge image rotated by 90° , 180° , and 270° . Rotation of the edge image has $O(N^2)$ complexity on $1 \times N$ processing arrays. The 180° and 270° rotations can be produced by further rotation of the 90° -rotated image or by flipping the 0° and 90° images in x and y . A rotation algorithm is illustrated in figure 8 and described in section 5.

In one version of the Hough transform [3, 6], each edge pixel in a binary image "votes" for all the straight lines that could intersect it. Votes are accumulated in a parameterized space (ρ, θ) , where $\rho = x \cos \theta + y \sin \theta$, (x, y) is the position of an edge pixel in the image, θ is the angle between the x axis and a normal to a hypothetical line in the image ($-90^\circ \leq \theta \leq 90^\circ$), and ρ is the orthogonal distance from the origin to that line. Straight lines can also be characterized by (v, θ) where v is the intercept of the line with one of the image borders (see figure 1a). A simple mapping exists between the (v, θ) space and the (ρ, θ) space. The algorithms in this paper characterize the Hough space in terms of v and θ .

The Radon transform takes a set of projections of an image taken from different angles and is computed by integrating the image intensities over paths defined by $\{(x, y) \mid \rho = x \cos \theta + y \sin \theta\}$. For the discrete case, the Hough transform is equivalent to the Radon transform applied to a binary image. The algorithms implemented in this paper compute the Radon transform since a binary edge map is first generated and then pixel values on particular paths are summed.

$O(NP)$ parallel algorithms for the Hough transform on SIMD $N \times N$ -array processors are described in [2] and [9]. The complexity can be reduced to $O(N + P)$ when independent addressing is available or the SIMD architecture is augmented with additional tree-structured processors [2]. Two $O(N + P)$ algorithms are described in [2] and [5] for simple mesh arrays. In both algorithms,

¹ P refers to the number of angles polled. Typically $P = 180$ for $-90^\circ \leq \theta \leq 90^\circ$. N is the number of rows and columns in the image.

columns (or rows) of Hough accumulators are pipelined through the image and follow paths based on the values of θ they are meant to accumulate. An $O(N + \sqrt{KP})$ algorithm is also presented in [5] that partitions the image into subimages, where K is the number of ρ -values being considered. Parallel Hough-transform algorithms have also been published for systolic arrays [1] and scanline array processors [4].

Section 2 of this paper briefly describes the AIS-5000 parallel vision processor. Sections 3 and 4 outline the two parallel algorithms implemented, and section 5 describes the rotation algorithm. Section 6 discusses the implementation of the algorithms on the AIS-5000. Section 7 concludes the paper.

2 The AIS-5000 Parallel Vision Processor

The AIS-5000 [8, 11, 12] is a commercially available $1 \times N$ -array SIMD vision processor used for a variety of applications in both industrial and research environments. As few as 128 or as many as 1024 bit-serial processors are arranged in a 1-dimensional linear array with a direct communication link between adjacent processors. Each processor has 32k memory associated with it in which an entire column of an image can be stored. The columns of memory can also be used to store alternate images or the results of intermediate computations. Grey-scale images are stored in separate bitplanes.

The architecture is anisotropic in that the horizontal dimension is massively parallel and the vertical dimension is similar to a standard single-processor architecture with random memory access. This allows a virtual 2-dimensional SIMD programming model to be used in image processing. A parallel-array controller is responsible for loading instructions into the array of processing elements. When processing of one row is completed, the parallel-memory addresses

can be incremented to process the next and remaining rows in the image. This architecture results in low overhead for loading parallel-processor instructions and greatly simplifies the data-communication channels between both real processors and virtual processors (which are assigned to different rows in the image).

Programming the AIS-5000 is accomplished through an extensive library of high and low-level image-processing functions that are invoked through the C programming language. These include arbitrary boolean operations on as many as four bitplanes, boolean operations using the four nearest neighbours of pixels in the same bitplane, and arithmetic operations using the carry register of each processor. As a default, the library functions operate on an entire image using the virtual 2-dimensional SIMD model described above. However, image processing can also be restricted to any subset of the image rows. This feature is taken advantage of in algorithm 2 where many of the bookkeeping operations need only be computed in a $1 \times N$ array.

3 Algorithm 1

This algorithm is based on the first algorithm in [2] and has a complexity of $O(N^2 + PN)$ on $1 \times N$ -array machines. The algorithm described in [2] has a complexity of $O(NP)$ on a simple $N \times N$ mesh array. This would correspond to an $O(N^2P)$ complexity on a $1 \times N$ processing array. However, we have improved the complexity on the AIS-5000 by taking advantage of random memory access within an image column. The comparable complexity of our algorithm on an $N \times N$ processing array would be $O(N + P)$.

The underlying strategy of algorithm 1 is to shear edge images (the rotated versions as well as the original edge image) in the x direction so that pixel values (0 or 1) can be summed in the vertical y direction.

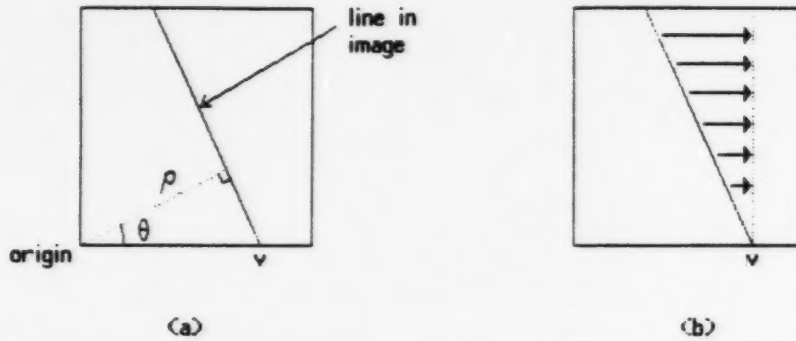


Figure 1: Lines in the edge image can be characterized by θ and ρ (figure 1a). They can also be characterized by θ and v , where v is intersection of the line with one of the image borders (the bottom border in this case). Figure 1b illustrates the row shifts in algorithm 1 prior to summing the columns.

Assume for the moment that $0^\circ \leq \theta \leq 45^\circ$ and θ is incremented by 1° at each iteration. For each θ value, each edge-image pixel is shifted right in x by a certain amount such that all pixels falling on a line with x -intercept v , at an angle of θ° , will line up vertically in the v th column². The columns are then summed (in parallel) to produce the Hough bin count for that particular θ and v (see figure 1b).

The shifting is done incrementally as follows. For $\theta = 0^\circ$ the columns are simply summed. The sums are transferred to a row in a $P \times N$ Hough-bin accumulator where the row coordinate of the accumulator represents θ and the column coordinate denotes the v value. Next, an $N \times N$ frame of v values (referred to as V) is computed for $\theta = 1^\circ$ with the formula³ $V = \rho / \cos \theta = X + Y \tan \theta$. For $\theta = 1^\circ$, the values in each row of V will indicate the distance in pixels that each edge-image row will be shifted in order for the correct pixels to line up in vertical columns. For succeeding iterations, the rows are shifted from their position on the previous iteration using the $N \times N$ frame ΔV_θ , where $\Delta V_\theta = V_\theta - V_{\theta-1^\circ}$.

²The pixels that are shifted out of the edge frame will not be column-summed and hence will not accumulate in the bin counts. They will, however, be counted when the process is repeated on the rotated edge images.

³The $N \times N$ X and Y frames are setup at the start by filling the X frame with the appropriate x coordinate in each column, and the Y frame with the appropriate y coordinate in each row. V is computed with floating-point accuracy.

The rows are shifted by creating a shift-mask S with 1's where $\Delta V_\theta > 0$ and 0's elsewhere⁴. Rows of S will contain either all 0's or all 1's. Rows are shifted 1 pixel at a time by setting E to $[(-S \cdot E) | (S \cdot E')]$, where E is the current edge image (sheared by previous iterations) and E' is E shifted right by 1 pixel. ΔV_θ is decremented and the process is repeated for $\lceil N(\tan \theta - \tan(\theta - 1^\circ)) \rceil$ iterations before the columns are summed to accumulate bin counts.

In addition to shifting the original (unrotated) edge image, ΔV_θ is also used to shift the edge frames rotated by 90° , 180° , and 270° . The Hough bins for the rotated images correspond to Hough bins for different θ and ρ values of the unrotated edge image. Figure 2 illustrates the bin counts accumulated by the rotated edge images⁵.

The Hough space that results is indexed by assigning a θ value to rows in the Hough-bin frame. The columns of the Hough-bin frame index the v values which can be assigned to the intercept of the line with the top, bottom, left, or right border of the original edge image. Values of ρ can be easily calculated from the column indices of the Hough space. Since the intercepts with the edge-image borders are used to index the Hough bins, the precision of the results (in terms of ρ) will vary depending on θ . For $\theta = 0^\circ$ or $\theta = \pm 90^\circ$, v and ρ values are identical and are in unit increments. For $\theta \neq 0^\circ$, values of ρ will be computed at increasing sub-unit accuracy as θ approaches $\pm 45^\circ$ (the minimum ρ increment is $\sim .7$ for $\theta = \pm 45^\circ$).

The algorithm can also be applied to edge images without requiring that the image be rotated. In this case, the original image is shifted left and right using the ΔV_θ shift-template, as before, to produce Hough bins for $-45^\circ \leq \theta \leq 45^\circ$. The Hough bins that would have resulted from the edge image rotated by 180° are accumulated by using the original edge image with a ΔV_θ shift-

⁴The shift-mask is created by OR'ing the bitplanes of ΔV_θ to locate non-zero values. The maximum value of ΔV_θ (for 1° increments) is $N(\tan 45^\circ - \tan 44^\circ)$, hence, only $\lceil \log\{N(\tan 45^\circ - \tan 44^\circ)\} \rceil$ bitplanes of ΔV_θ need to be OR'ed. The shift-mask is also used to decrement ΔV_θ after each 1-pixel shift.

⁵There is some overlap in the Hough bins in that some bin counts for θ and ρ will be accumulated twice. For example, the original image and the image rotated by 180° will both accumulate Hough bins for lines that intersect both the top and bottom of the image. This duplicate computation is essentially free since the processing is done in parallel.

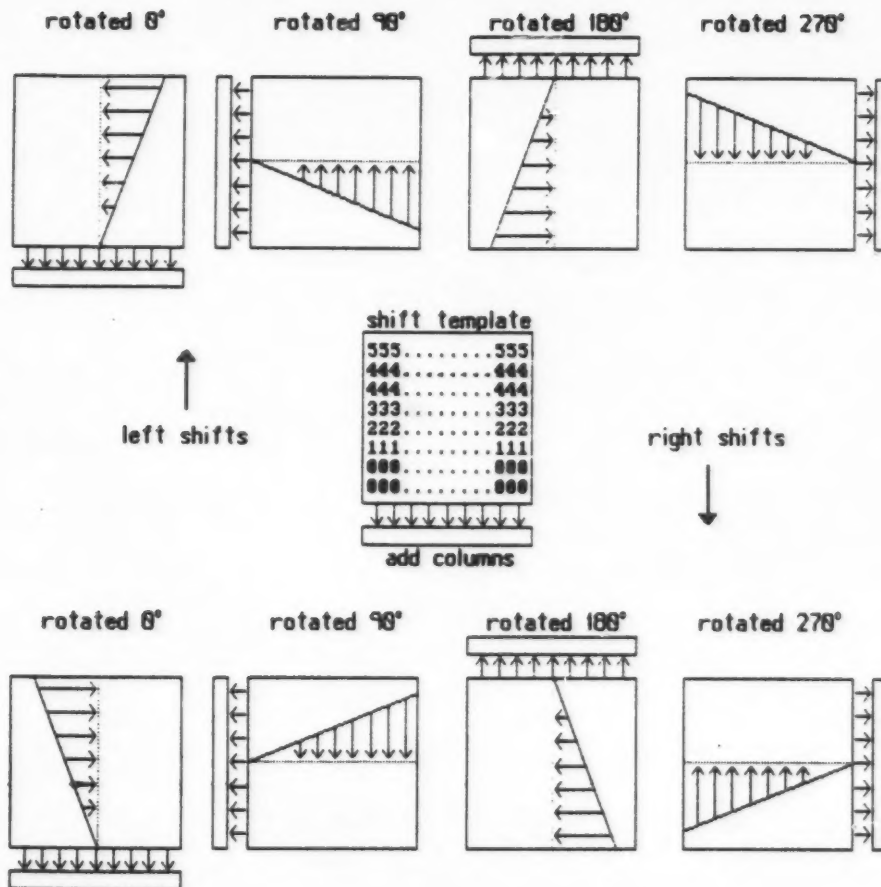


Figure 2: In algorithm 1, all the rotated edge images are shifted using the same shift-template. The figures at the top and bottom of the diagram illustrate the net result of shifting the rotated edge images in terms of operations on the unrotated edge image.

template computed from a Y frame in which the rows are inverted. Hence the bottom rather than the top of the image will be shifted in order to align the edge pixels prior to summing the columns. The Hough bins corresponding to the 90° and 270° rotated edge images are obtained by summing the *rows* after shifting the columns with shift-templates that compute the y -intercepts instead of the x -intercepts. However, summing the rows is an $O(N^2)$ operation on the AIS-5000 which would increase the overall complexity to $O(N^2 + PN^2)$.

4 Algorithm 2

This algorithm was developed by us and also implemented on the AIS-5000. The basic strategy is to pass the $N \times N$ edge image over an $N \times P$ frame of Hough bins (P is the number of angles polled) in such a way that the entire edge frame can be added to the Hough-bin frame at each iteration to accumulate bin counts. The columns of the edge frame are adjusted (sheared) at each iteration so that each edge pixel will eventually pass through all the Hough bins that it should increment. The Hough space is indexed by θ and v , where v is the intercept of the line with one of the image borders (see figure 3). Notice that there is a maximum count of 4 (in figure 3) at $\theta = 45^\circ$ and $v = N - 1$, and there are also a number of non-zero counts at other locations in the Hough space.

As in the first algorithm, only the processing for $45^\circ \leq \theta \leq 90^\circ$ will be illustrated in detail. The entire Hough space will, however, be computed when similar steps are carried out on the rotated edge images.

Consider first the unrotated edge image and the case for $45^\circ \leq \theta \leq 90^\circ$, where v is the y -intercept at the left edge of the image ($\rho = v \sin \theta$). The $N \times P$ Hough bin frame is arranged with θ indexed by the columns; $\theta = 90^\circ$ is the rightmost column and $\theta = 45^\circ$ is the leftmost

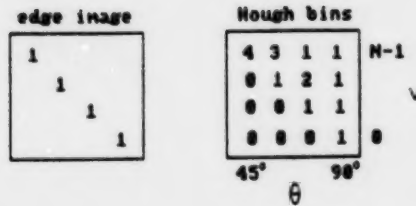


Figure 3: A simple 45° line and the Hough-bin frame in algorithm 2 after the edge image has been passed over it. These Hough bins accumulate the counts for $45^\circ \leq \theta \leq 90^\circ$. v refers to the intercepts of hypothetical lines with the left border of the image.

column. For the current implementation, $P = N$ for a range of 45° θ values (e.g. $45^\circ \leq \theta \leq 90^\circ$), i.e. there are N angle bins and $N - 1$ angle increments. We chose $P = N$ just for illustration. P can be chosen independent of N , for example, $P = 45$ for 1° -increments. The angle referenced by the Hough bin at column i is $\tan^{-1}(\frac{N}{N-i})$, where $0 \leq i < N$. The v -intercept values form the Hough-bin row indices.

Initially the edge image is aligned with its leftmost column over the rightmost column ($\theta = 90^\circ$) of the Hough-bin frame. For each of $2N$ iterations (or $N + P$, generally speaking), the edge image is moved left by 1 pixel and added to the stationary Hough-bin frame. Each column of the edge image is optionally shifted up 1 pixel at each iteration to ensure that the pixel values are added to the appropriate Hough bins. For example, for the rightmost column of the Hough-bin frame ($\theta = 90^\circ$, which accumulates bin counts for horizontal lines) all the edge-image pixels will pass through these bins without any column shifts. For the bins in the leftmost column of the Hough-bin frame ($\theta = 45^\circ$), column i of the edge image will have been shifted up i pixels as that column passes over the leftmost column of the Hough-bin frame. Figure 4 illustrates the column shifts at various stages of the algorithm.

A system of $1 \times N$ bookkeeping arrays is used to ensure that the correct amount of edge-image column shifting occurs as the edge image passes over the Hough-bin frame. Some of the bookkeeping arrays move with the edge image and some are kept stationary with the Hough-bin

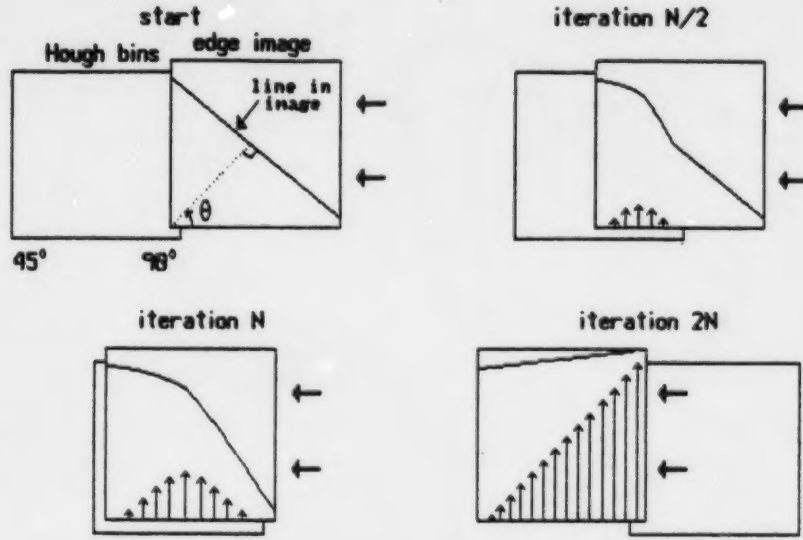


Figure 4: The edge-image column shifts at various stages in algorithm 2 for the case when $45^\circ \leq \theta \leq 90^\circ$. The edge image is added to the Hough bin frame at each iteration.

frame. The cumulative amount of column-shifting undergone by each column of the edge image and the amount of column-shift required for each Hough-bin column is updated at each iteration.

The following frame symbols are used to describe the algorithm. The frame subscript indicates the size of the frame (either $1 \times N$ or $N \times N$) and a frame argument indicates a column number; for example, $M_{1 \times N}(i)$ refers to the i th column of the $1 \times N$ frame M , where $0 \leq i < N$. Figure 5 illustrates the various frames used and their initial values.

- $E_{N \times N}$ is the 1-bitplane edge image.
- $B_{N \times N}$ is the Hough bin frame. The rows index v and the columns index θ . $B_{N \times N}$ has $\log N$ bitplanes.
- $SE_{1 \times N}$ records the number of vertical shifts each column of the edge image has undergone.
- $XE_{1 \times N}$ stores the x coordinates of the edge image. $XE_{1 \times N}(i) = i$.
- $SB_{1 \times N}$ records the number of vertical shifts each Hough bin requires.

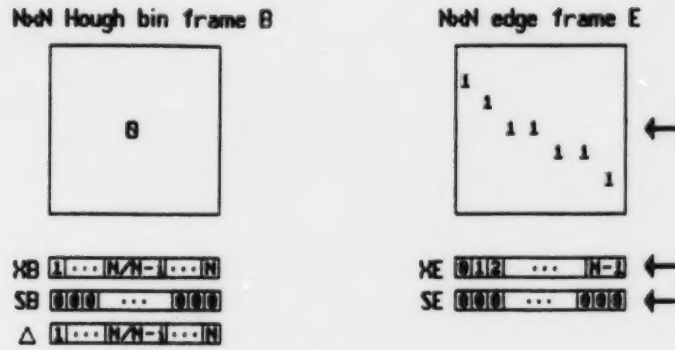


Figure 5: The configuration of the $1 \times N$ bookkeeping frames at the start of algorithm 2. The $XE_{1 \times N}$ and $SE_{1 \times N}$ frames are shifted left 1 pixel at each iteration along with the edge-image frame $E_{N \times N}$. The $XB_{1 \times N}$, $\Delta_{1 \times N}$, and $SB_{1 \times N}$ frames remain stationary along with the Hough-bin frame $B_{N \times N}$. i refers to the column number.

- $XB_{1 \times N}$. Column i of $XB_{1 \times N}$ records the x coordinate where the next vertical shift of the edge image should occur for the Hough bin at column i . $XB_{1 \times N}$ is initially set to $\Delta_{1 \times N}$.
- $\Delta_{1 \times N}$, where $\Delta_{1 \times N}(i) = \frac{N}{N-i}$ for $0 \leq i < N$. The value in column i of $\Delta_{1 \times N}$ represents the number of horizontal shifts⁶ that the edge image must undergo before a 1-pixel vertical shift is required for the Hough bin in column i . For example, column 0 of $\Delta_{1 \times N}$ (corresponding to the $\theta = 45^\circ$ Hough bin) has the value 1 which indicates that for every horizontal shift of the edge image, the columns must be shifted vertically by 1 pixel. Column $N - 1$ of $\Delta_{1 \times N}$ (corresponding to the $\theta = 90^\circ$ Hough bin) has the value N which indicates that N horizontal shifts of the edge image are required before a 1-pixel vertical shift will occur—in other words no vertical shift of the image is required, hence, horizontal-line pixels will accumulate in column $N - 1$ of the Hough-bin frame.
- $M_{1 \times N}$ is a shift mask that indicates which columns of the edge image will be shifted vertically.

The frames $XE_{1 \times N}$ and $SE_{1 \times N}$ are shifted over the Hough-bin frame along with $E_{N \times N}$. All the rest of the frames are stationary. The algorithm is described in detail in figure 6. The columns of

⁶ $\Delta_{1 \times N}$ is computed with floating point accuracy.

Initial setup:

Set the entire Hough-bin frame $B_{N \times N}$ to 0.

Set $SA_{1 \times N}(i)$ and $SB_{1 \times N}(i)$ to 0, for $0 \leq i < N$.

Set $XE_{1 \times N}(i)$ to i , for $0 \leq i < N$.

Set $\Delta_{1 \times N}(i)$ and $XB_{1 \times N}(i)$ to $\frac{N}{N-i}$, for $0 \leq i < N$.

Shift $E_{N \times N}$, $XE_{1 \times N}$, and $SE_{1 \times N}$ right by N pixels.

For $2N$ iterations do:

Shift $E_{N \times N}$, $XE_{1 \times N}$, and $SE_{1 \times N}$ left by 1 pixel.

Set $M_{1 \times N}(i)$ to 1 iff $XE_{1 \times N}(i) \leq XB_{1 \times N}(i) < XE_{1 \times N}(i+1)$, for $0 \leq i < N$.

Where $M_{1 \times N}(i)$ is 1, set $XB_{1 \times N}(i)$ to $XB_{1 \times N}(i) + \Delta_{1 \times N}(i)$, for $0 \leq i < N$.

Where $M_{1 \times N}(i)$ is 1, increment $SB_{1 \times N}(i)$, for $0 \leq i < N$.

Set $M_{1 \times N}(i)$ to 1 iff $SB_{1 \times N}(i) > SE_{1 \times N}(i)$, for $0 \leq i < N$.

Where $M_{1 \times N}(i)$ is 1, shift column i of $E_{N \times N}$ up by 1 pixel, for $0 \leq i < N$.

Where $M_{1 \times N}(i)$ is 1, increment $SE_{1 \times N}(i)$, for $0 \leq i < N$.

Set $B_{N \times N}$ to $B_{N \times N} + E_{N \times N}$.

end.

Figure 6: Algorithm 2 for the case when $45^\circ \leq \theta \leq 90^\circ$. All other cases are similar except that a different set of Hough bins and edge images (rotated by 0° , 90° , 180° , or 270°) are used and edge-image columns may be shifted down instead of up. References to column position i are with respect to the stationary frames.

the edge image are shifted with the same method used in algorithm 1 (see section 3). However, for algorithm 2 the shift-mask must be transferred from the the single-row frame $M_{1 \times N}$ to a full frame shift-mask. Figure 7 shows the detailed stages of the algorithm for a very small image.

Cases other than $45^\circ \leq \theta \leq 90^\circ$ are dealt with in a manner analogous to algorithm 1. For the case when $-90^\circ \leq \theta \leq -45^\circ$ another copy of the unrotated edge image is shifted over a different $N \times N$ frame of Hough bins using the same shift-mask $M_{1 \times N}$. For this case, however, the edge-frame columns will be optionally shifted down instead of up at each iteration. The cases for $-45^\circ \leq \theta \leq 0^\circ$ and $0^\circ \leq \theta \leq 45^\circ$ are computed with two copies of the edge image rotated by 90° (also using the same shift-mask). As in the first algorithm, the 180° -rotated edge image will accumulate bin counts for lines (for $\theta \leq -45^\circ$ or $\theta \geq 45^\circ$) that intersect the right border of the image but not the left border, and the 270° -rotated edge image will accumulate bin counts

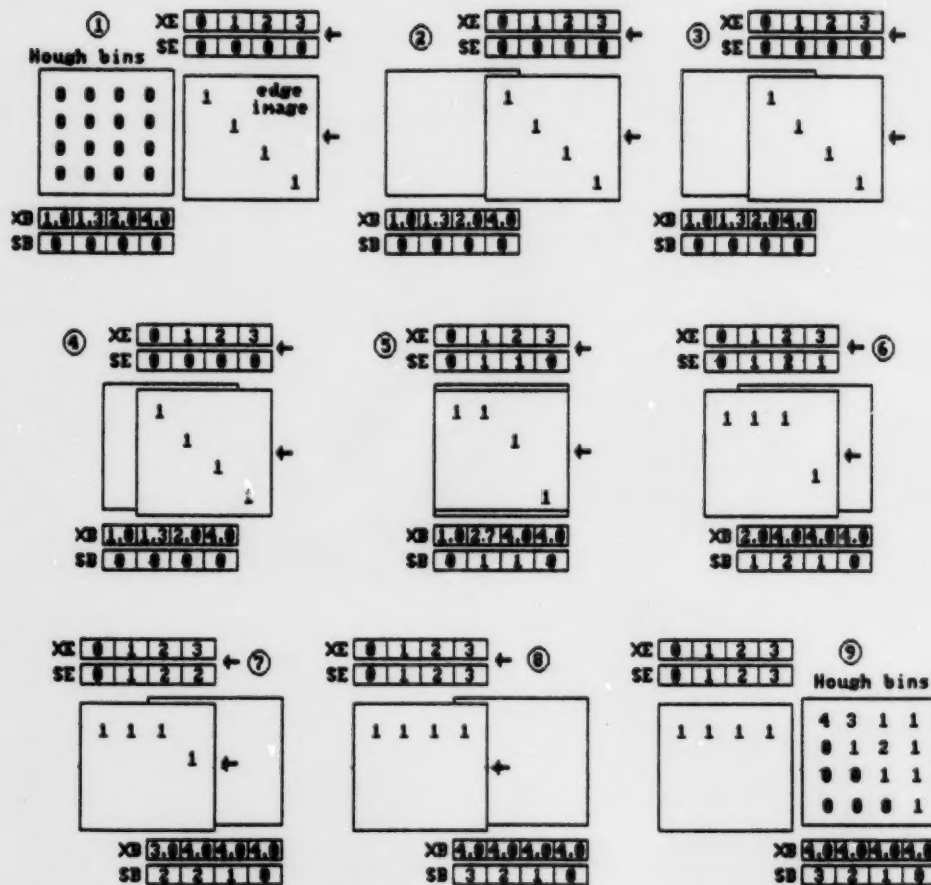


Figure 7: A detailed example of the bookkeeping arrays in algorithm 2 for a simple 45° line in a small 4x4 image. At each iteration the edge image is added to the Hough-bin frame.

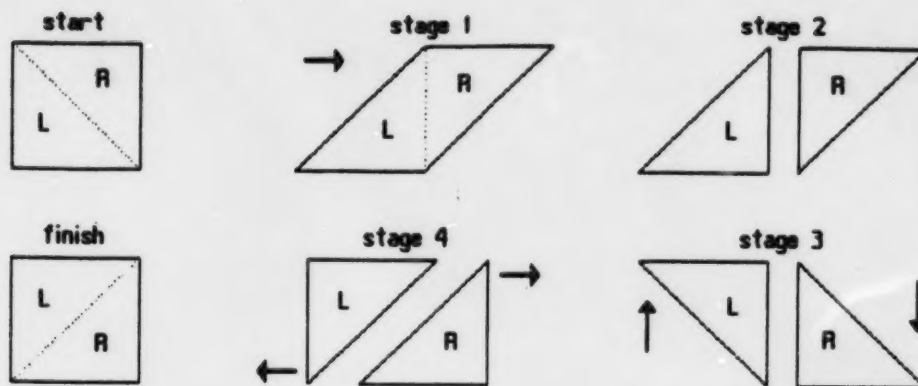


Figure 8: The stages of an algorithm to rotate an image by 90° .

for lines (for $-45^\circ \leq \theta \leq 45^\circ$) that intersect the top border of the image but not the bottom border⁷.

5 Rotation Algorithm

As indicated by the previous sections, it is sometimes necessary to rotate an edge image. The image rotation can be accomplished on the AIS-5000 by a series of row and column shearing operations. An $N \times N$ frame of x and y coordinates (referred to as X and Y) is created and used as a shift-template. In the first stage (see figure 8) the image rows are sheared 1 pixel to the right⁸ wherever $Y > 0$. Y is decremented and the process repeated for N iterations. Rows are shifted by the method described in algorithm 1 (see section 3). The image is then split into left and right halves. The columns of the right half are sheared down using X as a shift-mask. The left half is sheared up⁹ wherever $X < N - 1$ and X is incremented at each iteration (stage 3 in

⁷The bin counts accumulated with the 180° and 270° -rotated images could also be collected by passing the unrotated and 90° -rotated edge images over a set of Hough bins from the left instead of from the right.

⁸The shift-mask is created by OR'ing the bitplanes of Y . The shift-mask is also used to decrement Y at each iteration.

⁹The shift-mask is created by AND'ing the bitplanes of X and inverting the result. The shift-mask is used to increment X at each iteration.

figure 8). The rows of the right half are then sheared right wherever $Y > 0$ and the rows of the left half are shifted left wherever $Y < N - 1$ (stage 4 in figure 8). Combining the left and right halves results in the image rotated by 90° .

6 Implementation on the AIS-5000

6.1 Complexity

Algorithm 1 first shifts the edge image along the rows to get the edge elements aligned in columns, and the shifting is executed incrementally for all the angles. Therefore, the shifting process has a complexity of $O(N)$ on an $N \times N$ mesh array and a complexity of $O(N^2)$ on the AIS-5000 which is a $1 \times N$ array processor. Second, the summation along the columns takes only $O(N)$ steps for each θ value on the AIS-5000, because the algorithm takes advantage of random memory access within an image column. This yields a complexity of $O(PN)$ for all P different angles. The overall complexity for Algorithm 1 is hence $O(N^2 + PN)$.

Algorithm 2 takes $O(N + P)$ iterations to shift the edge image row-wise over the Hough bins. At the same time, selected columns are shifted column-wise to bring the edge elements to the appropriate Hough bins for accumulation. Similar to Algorithm 1, the column shifts are executed incrementally for all the angles and, thus, take $O(N)$ steps altogether. The bin count accumulation requires $O(N + P)$ operations as the edge elements are accumulated to their respective Hough bins at every iteration. The overall process, therefore, has a complexity of $O(N + P)$ for an $N \times N$ mesh array and a complexity of $O(N^2 + PN)$ for the AIS-5000.

A more detailed analysis can be obtained by counting the number of operations¹⁰ on $N \times N$

¹⁰An operation is considered to be a single C-library function applied to an $N \times N$ frame. Each operation is $O(N)$ on a $1 \times N$ array processor since each row is processed separately. Unfortunately, a more detailed breakdown

frames, and the number of operations on single rows. Algorithm 1 requires $34N+8P$ operations on $N \times N$ frames and $16P$ operations on single rows. Since each operation on an $N \times N$ frame is $O(N)$, approximately $34N^2 + 8PN + 16P$ total operations are required for algorithm 1. Algorithm 2 requires $52N$ operations on $N \times N$ frames and $96N$ operations on single rows for a total of $52N^2 + 96N$ operations for the current implementation where $P = 4N$ for the full θ range of 180° .

6.2 Experimental Results

We have built a working system with the AIS-5000 as the major processing unit and a SUN-3 workstation as the host. Live images are taken from a camera and processed in near-real time. The edge detection process takes about 20 ms. The execution time for algorithm 1 is 2.0 seconds on 256×256 images when generating a relatively high resolution Hough space (256×180) using a θ -increment of 1° , $P = 180$. Algorithm 2 takes a little longer in the current implementation, because θ is at a very high resolution, i.e. $P = 256$ for each 45° range, and the total resolution of θ for the full 180° range is 1024. In fact, the angle increments for P in such a rate provide more θ -resolution than is usually desired. Future implementations of algorithm 2 will allow any size of θ -resolution—for example, 1° increments—and we expect a significant reduction in execution time. Figures 9 and 10 illustrates the Hough space resulting from real and artificially created edge images. Figure 11 illustrates the edge image at various stages in algorithm 2.

6.3 System Limitations

The amount of parallel memory available on the AIS-5000 presented a problem for both algorithms. The machine used to implement the algorithms had 512 processors with 32k of 1-bit memory for each processor. With the system configured for 512×512 images, 64 bitplanes were of the number of steps involved in each high-level C function is not available.

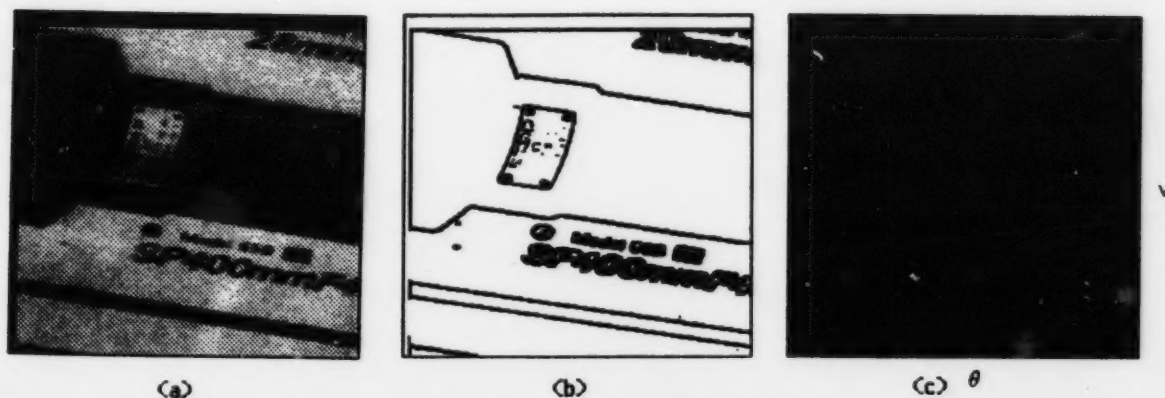


Figure 9: The Hough space for a real image. Figure 9a is the original 8-bit image, figure 9b is the binary edge image extracted from it, and figure 9c is the Hough space for angles in the range $45^\circ \leq \theta \leq 90^\circ$, where v is the y -intercept.

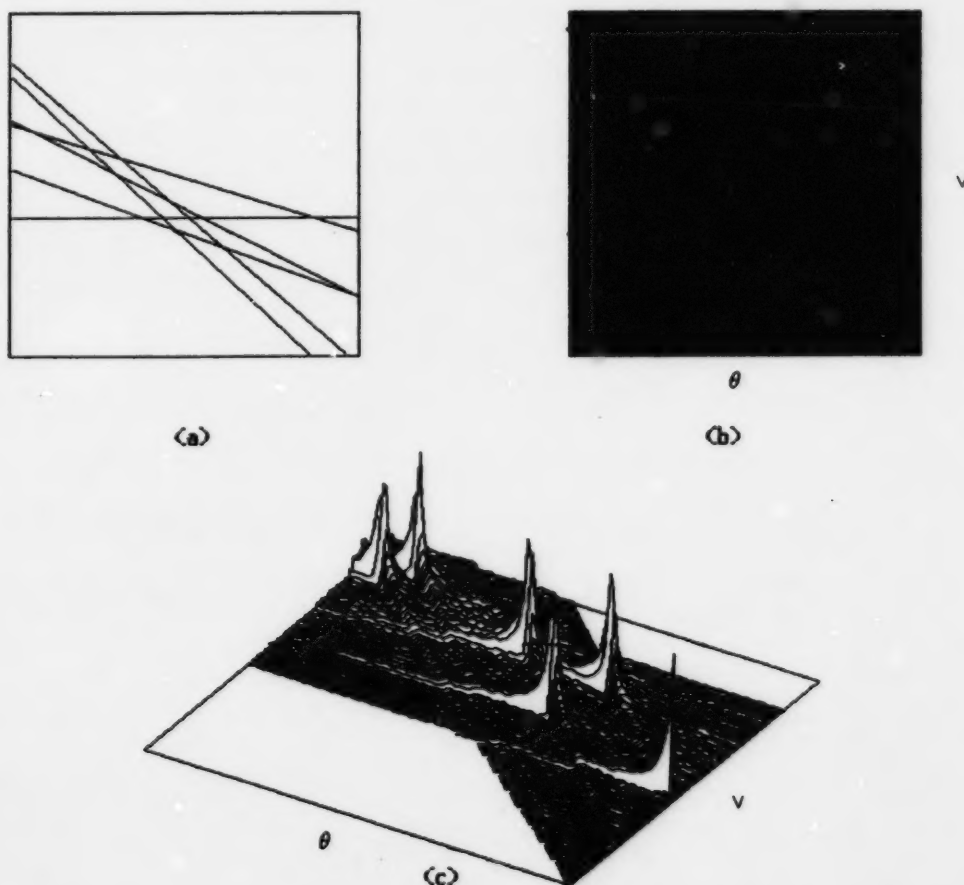


Figure 10: The Hough space for an artificially created edge image. Figure 10a is the artificially generated edge image. Figure 10b shows the Hough space for $45^\circ \leq \theta \leq 90^\circ$ where v is the y -intercept and the brightness depicts the accumulated count of each Hough bin. Figure 10c shows a "3-D" representation of it.

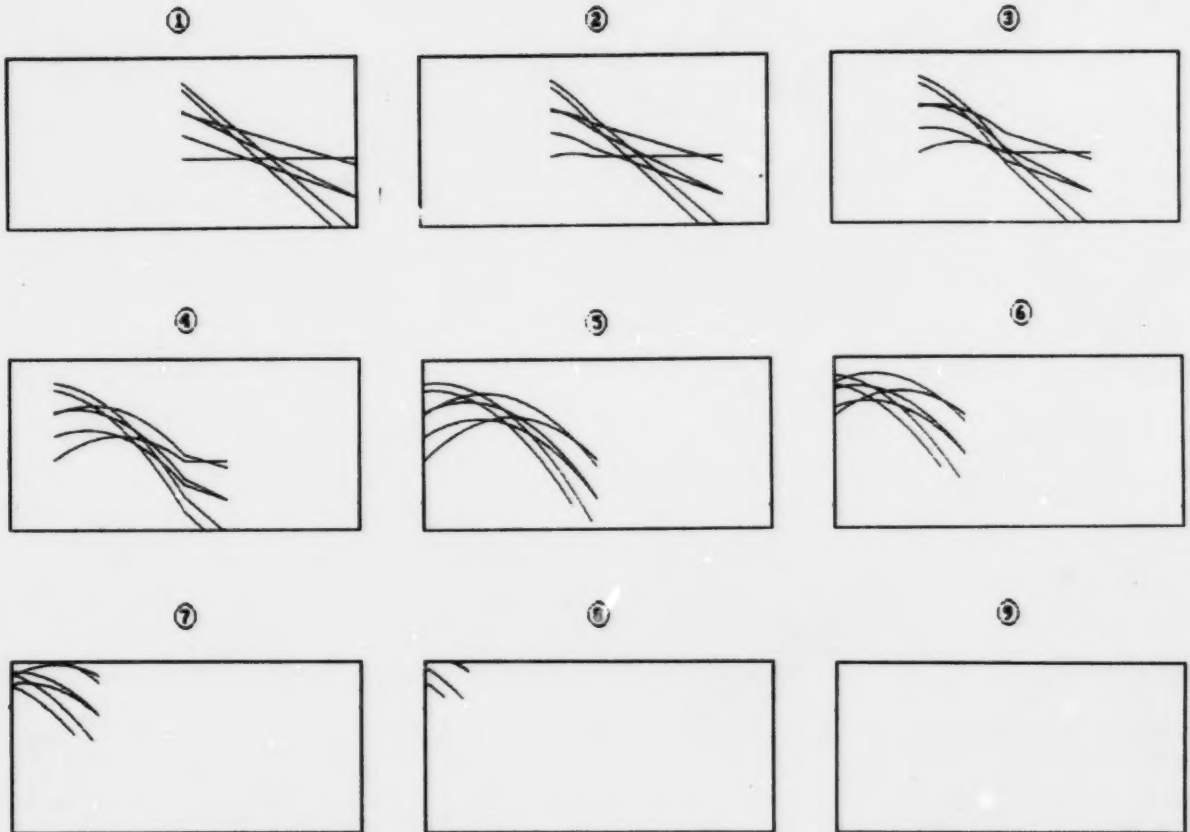


Figure 11: The edge image at various stages in algorithm 2. Selected columns of the edge image are shifted up as it passes over the Hough bins (located in the left half of the frames). The Hough space illustrated here is for $45^\circ \leq \theta \leq 90^\circ$.

available for storing images and intermediate computations. This was insufficient for both algorithms. Configuring the system for 256x256 images increased the available memory which allowed algorithm 1 to execute for 1° increments. However, there was not enough memory to store the complete Hough space for algorithm 2. For this case, the Hough spaces would have to off-loaded from the parallel memory to make additional storage available during the execution of the algorithms. The principal memory requirement (aside from storage for the Hough bins) is for scratch memory used in the calculation of the V frames in algorithm 1. The results of arithmetic computations stored in these frames were scaled up by a factor of N in order to compute values with sufficiently high accuracy.

In algorithm 2 the reduction in execution time for processing $1 \times N$ bookkeeping arrays on the AIS-5000 was not as much as anticipated. In certain situations, the overhead of loading the processing elements with instructions overshadows the reduced computations resulting from restricting the number of rows processed. This increased the execution time of algorithm 2 by a significant amount.

The use of $\frac{N}{2} \times \frac{N}{2}$ images in the algorithms (where N is the number of processors) was used to advantage in algorithm 2 since the edge images were initially shifted right by an amount equal to the image width. For an image width of N , this shifting would have resulted in the images being shifted entirely out of parallel memory. An image width of $\frac{N}{2}$ allows the edge image and Hough-bin frame to co-exist side by side at the start of algorithm 2. For processing of $N \times N$ images, algorithm 2 could be modified to perform the Hough-bin accumulation in two stages by computing the left and right halves of the Hough space separately.

Rotating the edge image by 90° , 180° , and 270° , using the algorithm described in section 5, took approximately 3 seconds on the AIS-5000 parallel vision computer. Image rotation requires global operations and is not particularly suited to implementation on mesh arrays or 1-

dimensional array machines. In fact, image rotation on a single-processor machine has the same $O(N^2)$ complexity as the algorithm described above. One solution might be to obtain rotated images when the digitized image is first scanned into memory using special-purpose hardware.

7 Conclusion

There has been a lot of interest recently in parallel algorithms for the Hough transform. In many cases the algorithms have been specified for hypothetical machines without being implemented on real hardware. We have chosen to develop algorithms on real machines.

Two algorithms for computing the Hough transform for straight lines on $1 \times N$ processing arrays have been presented. Algorithm 1 is adapted from the first algorithm in the paper by Cypher et al. [2]. It has been improved from an $O(NP)$ algorithm to $O(N + P)$ on an $N \times N$ mesh and $O(N^2 + PN)$ on our $1 \times N$ array processor. We have also developed a new algorithm with the same complexity. Both algorithms have been implemented on the AIS-5000 parallel vision computer which has 512 1-bit SIMD processors connected as a 1-D array [8, 11, 12]. Both algorithms have a complexity of $O(N^2 + PN)$, where P is the number of angles polled. This complexity is within $O(N)$ of the optimal $O(N + P)$ complexity reported for algorithms for $N \times N$ mesh arrays [2, 5].

The AIS-5000 supports a virtual 2-dimensional $N \times N$ programming model and the algorithms have, for the most part, been described in those terms. Both algorithms could be easily adapted to run on mesh-array computers. (The second algorithm would compute the same bookkeeping information in all rows instead of just one row.) The run-times of the algorithms on the AIS-5000 are on the order of 2 seconds for $N = 256$ when high resolution Hough spaces are generated. It is reasonable to expect an $O(N)$ speed increase on $N \times N$ mesh arrays, resulting in run-times

approximately 20–30 ms for 256x256 images. These times compare well with the reported times for the Hough transform on a variety of parallel computers [7].

References

- [1] Henry Y. H. Chuang and C. C. Li. A systolic array processor for straight line detection by modified Hough transform. In *Proceedings of the IEEE Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, pages 300–304, Miami Beach, Florida, Nov. 18–20 1985.
- [2] R.E. Cypher, J.L.C. Sanz, and L. Snyder. The Hough transform has $O(N)$ complexity on SIMD $N \times N$ array architectures. In *Proceedings of the IEEE Workshop on Computer Architecture for Pattern Analysis and Machine Intelligence*, pages 115–121, Seattle, Washington, Oct. 5–7 1987.
- [3] Frank Evans. A survey and comparison of the Hough transform. In *Proceedings of the IEEE Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, pages 378–380, Miami Beach, Florida, Nov. 18–20 1985.
- [4] Allan L. Fisher and Peter T. Highnam. Computing the Hough transform on a scan line array processor. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(3):262–265, March 1989.
- [5] Concettina Guerra and Susanne Hambrusch. Parallel algorithms for line detection on a mesh. *Journal of Parallel and Distributed Computing*, 6:1–19, 1989.
- [6] J. Illingworth and J. Kittler. A survey of the Hough transform. *Computer Vision, Graphics, and Image Processing*, 44:87–116, 1988.

- [7] Azriel Rosenfeld. A report on the DARPA image understanding architectures workshop. In *Proceedings: Image Understanding Workshop, Vol. 1*, pages 298-302, Los Angeles, California, Feb. 23-25 1987. Defense Advanced Research Projects Agency.
- [8] Lorenz A. Schmitt and Stephen S. Wilson. The AIS-5000 parallel processor. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(3):320-330, May 1988.
- [9] Teresa M. Silberberg. The Hough transform on the geometric arithmetic parallel processor. In *Proceedings of the IEEE Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, pages 387-393, Miami Beach, Florida, Nov. 18-20 1985.
- [10] Bruce W. Suter. A parallel algorithm for the discrete radon transform. In *Proceedings of the IEEE Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, pages 394-398, Miami Beach, Florida, Nov. 18-20 1985.
- [11] Stephen S. Wilson. One dimensional SIMD architectures: The AIS-5000. In S. Levialdi, editor, *Multicomputer Vision*, chapter 9, pages 131-149. Academic Press, 1988.
- [12] Stephen S. Wilson. The Pixie-5000: A systolic array processor. In *Proceedings of the IEEE Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, pages 477-483, Miami Beach, Florida, Nov. 18-20 1985.

Centre for Systems Science

- Simon Fraser University established the **Centre for Systems Science** as a provincial centre for advanced study. The objective of CSS is twofold: coordinate and combine advanced systems research at SFU, and encourage application of research results within industry in order to further the technological development of British Columbia.
- CSS has assembled three research groups to streamline its efforts. The orientation of these groups is as follows: intelligent systems, microelectronics, and computer and communications systems. Because of shared research interests, this latter group merges with LCCR (**Laboratory for Computer and Communications Research**) for the publication of technical reports.
- CSS's three groups involve over 80 skilled faculty researchers principally from SFU's Schools of Engineering Science and Computing Science, but also draw substantially on Mathematics, Physics, Geography, Philosophy, Linguistics, Psychology, Communication, and Kinesiology.